

Министерство образования Республики Беларусь
Учреждение Образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра электронных вычислительных средств

Лабораторная работа № 7
«ИЗУЧЕНИЕ АЛГОРИТМОВ РАЗМЕЩЕНИЯ ЭЛЕМЕНТОВ»

Выполнили:
ст. гр. 850702
Маковский Р. А.
Турко В. Д.

Проверил:
Станкевич А. В.

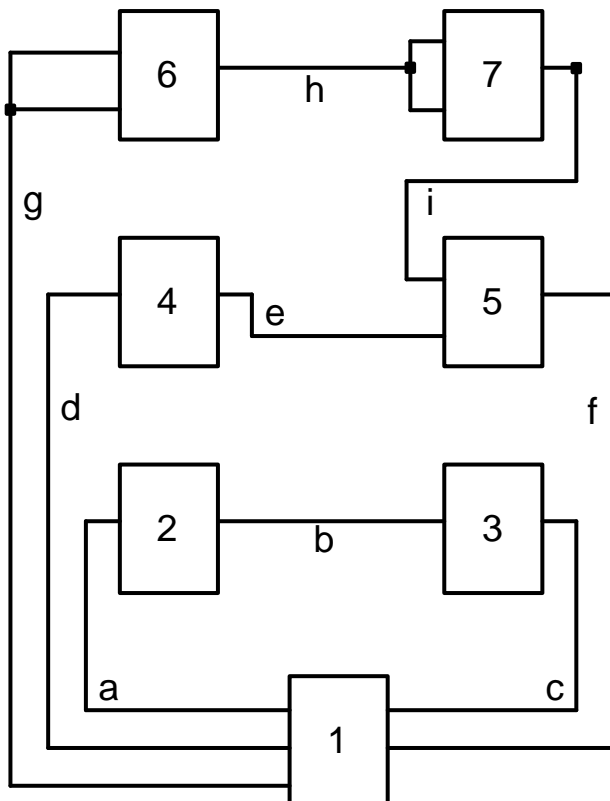
Минск 2020

ЦЕЛЬ РАБОТЫ:

Изучить алгоритмы размещения конструктивных элементов на печатной плате.

ИСХОДНЫЕ ДАННЫЕ:

В качестве соединителя выбран элемент 1, который должен быть расположен в позиции 1. Остальные элементы имеют однотипные корпуса и могут быть размещены в любой из оставшихся позиций. Значения весов всех цепей равны 100. В качестве критерия оптимизации использовали минимум суммарной взвешенной длины соединений.



5	6	7
2	3	4
-	1	-

Схема соединений элементов и расположение посадочных мест на печатной плате

```
Matrix POSITIONS = Matrix([[5, 6, 7], [2, 3, 4], [null, 1, null]]);
```

```
const Line A = Line('a', 2);  
const Line B = Line('b', 2);  
const Line C = Line('c', 2);  
const Line D = Line('d', 2);  
const Line E = Line('e', 2);  
const Line F = Line('f', 2);  
const Line G = Line('g', 3);  
const Line H = Line('h', 3);  
const Line I = Line('i', 2);
```

```
List<Element> ELEMENTS = [  
  Element('1', [A, C, D, F, G]),  
  Element('2', [A, B]),  
  Element('3', [B, C]),  
  Element('4', [D, E]),  
  Element('5', [E, F, I]),  
  Element('6', [G, H]),  
  Element('7', [H, I]);
```

ХОД РАБОТЫ:

1. Алгоритм случайного поиска.

Функция, реализующая генерацию одного размещения:

```
Attempt _random(List<Element> elements) {
    double sum = 0;
    List<int> positions = [2, 3, 4, 5, 6, 7];
    positions.shuffle();
    positions.insert(0, 1);
    for (int i = 0; i < elements.length; i++) {
        for (int j = 0; j < elements.length; j++) {
            final coordOfCurrent = POSITIONS.indexOf(positions[i]);
            final coordOfConnecting = POSITIONS.indexOf(positions[j]);
            sum += elements[i].connectionWeightTo(elements[j]) * coordOfCurrent.rela-
tiveLengthTo(coordOfConnecting);
        }
    }
    return Attempt(positions, sum);
}
```

Результат генерации 100 размещений:

	Лучший результат	Худший результат
Размещение элементов (элемент → позиция)	1 -> 1 2 -> 7 3 -> 4 4 -> 3 5 -> 2 6 -> 6 7 -> 5	1 -> 1 2 -> 2 3 -> 7 4 -> 4 5 -> 5 6 -> 6 7 -> 3
Суммарная взвешенная длина соединений	1300.0	1999.9

2. Алгоритм последовательного поиска.

Функция, генерирующая матрицу расстояний коммутационного поля платы:

```
Matrix _stateMatrix() {
    Matrix matrix;
    POSITIONS.forEach((item) {
        if (item != null) {
            List<int> lengths = [];
            POSITIONS.forEach((comparing) {
                if (comparing != null) {
```

```

        final coordOfCurrent = POSITIONS.indexOf(item);
        final coordOfConnecting = POSITIONS.indexOf(comparing);
        final result = coordOfCurrent.relativeLengthTo(coordOfConnecting);
        lengths.add(result);
    }));
    if (matrix == null) {
        matrix = Matrix([lengths]);
    } else {
        matrix.addRow(lengths);
    }));
    return matrix;
}

```

Функция, генерирующая матрицу взвешенной связанности:

```

Matrix _lengthsMatrix() {
    Matrix matrix;
    ELEMENTS.forEach((item) {
        if (item != null) {
            List<double> lengths = [];
            ELEMENTS.forEach((comparing) {
                if (comparing != null) {
                    if (item == comparing) {
                        lengths.add(0);
                    } else {
                        final result = item.connectionWeightTo(comparing);
                        lengths.add(result);
                    }
                }
            });
            if (matrix == null) {
                matrix = Matrix([lengths]);
            } else {
                matrix.addRow(lengths);
            }
        }
    });
    return matrix;
}

```

Функция, генерирующая матрицу расстояний коммутационного поля платы:

```

Attempt result(List<Element> elements, List<int> positions) {
    double sum = 0;
    for (int i = 0; i < elements.length; i++) {
        for (int j = 0; j < elements.length; j++) {
            final coordOfCurrent = POSITIONS.indexOf(positions[i]);
            final coordOfConnecting = POSITIONS.indexOf(positions[j]);
            sum += elements[i].connectionWeightTo(elements[j]) * coordOfCurrent.relativeLengthTo(coordOfConnecting);
        }
    }
    return Attempt(positions, sum);
}

```

Составим матрицу относительных расстояний

Позиция	5	6	7	2	3	4	1	Сумма
5	0	1	2	1	2	3	3	12
6	1	0	1	2	1	2	2	9
7	2	1	0	3	2	1	3	12
2	1	2	3	0	1	2	2	11
3	2	1	2	1	0	1	1	8
4	3	2	1	2	1	0	2	11
1	3	2	3	2	1	2	0	13

Наименьшая сумма относительных расстояний достигается при использовании вектора очередности позиций [3, 1, 2, 4, 6, 5, 7].

Составим матрицу взвешенной связанности

Позиция	1	2	3	4	5	6	7	Сумма
1	0.0	50.0	50.0	50.0	50.0	33.3	0.0	233.3
2	50.0	0.0	50.0	0.0	0.0	0.0	0.0	100.0
3	50.0	50.0	0.0	0.0	50.0	0.0	33.3	100.0
4	50.0	0.0	0.0	0.0	50.0	0.0	0.0	100.0
5	50.0	0.0	0.0	50.0	0.0	0.0	50.0	150.0
6	33.3	0.0	0.0	0.0	0.0	0.0	33.3	66.7
7	0.0	0.0	0.0	0.0	50.0	33.3	0.0	83.3

После сортировки по убыванию суммы получим вектор очередности размещения элементов [6, 2, 1, 3, 4, 5, 7].

Т.к. по условию необходимо разместить элемент 1 на позиции 1 в векторе размещения элементов поменяем местами 1 и 2 элементы, обладающие равными коэффициентами взвешенной связанности.

	Результат
Размещение элементов (элемент → позиция)	6 → 3 1 → 1 2 → 2 3 → 4 4 → 6 5 → 5 7 → 7
Суммарная взвешенная длина соединений	1600.0

ВЫВОД:

В лабораторной работе мы познакомились с алгоритмами случайного и последовательного поиска размещения конструктивных элементов на печатной плате, а также реализовали решение задачи размещения в ортогональной метрике по критерию минимума суммарной взвешенной длины соединений. В результате решения алгоритм случайного поиска дал лучший результат (меньшую суммарную длину), чем алгоритм последовательного поиска, однако при большем количестве элементов предпочтительнее использовать алгоритм последовательного поиска.